# A PYTHON PROGRAM TO IMPLEMENT K-MEANS MODEL

**Ex.No.: 9B**

**Date of Experiment: 24/10/2024**

**AIM:-**

To implement a python program using a K-Means Algorithm in a model.

**ALGORITHM:-**

Step1: Import all the other necessary libraries(numpy as np, matplotlib.pyplot as plt and sklearn.tree,pandas as pd and seaborn as sns).

Step2: Select the number K to decide the number of clusters.

Step3: Select random K points or centroids. (It can be different from the input dataset). Step4:

Assign each data point to their closest centroid, which will form the predefined K clusters. Step5:

Calculate the variance and place a new centroid of each cluster.

Step6: Repeat the fourth steps, which means assign each datapoint to the new closest centroid of each cluster.

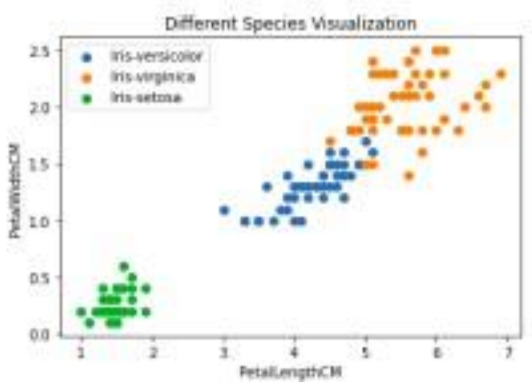
Step7: If any reassignment occurs, then go to step-5 else go to FINISH.

Step8: Train the model and plot the graph using scatterplot() function.

**IMPLEMENTATION:-**

```
data = pd.read_csv('../input/k-means-clustering/KNN (3).csv')
data.head(5)
```

```
Text(0.5, 1.0, 'Different Species Visualization')
```




```
req_data = data.iloc[:,1:]
req_data.head(5)
```

| | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|---|---|---|---|---|
| 0 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| 1 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| 2 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| 3 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| 4 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

```
shuffle_index = np.random.permutation(req_data.shape[0]) #shuffling the row index of our dataset
req_data = req_data.iloc[shuffle_index]
req_data.head(5)
```

| | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|---|---|---|---|---|
| 45 | 4.8 | 3.0 | 1.4 | 0.3 | Iris-setosa |
| 50 | 7.0 | 3.2 | 4.7 | 1.4 | Iris-versicolor |
| 135 | 7.7 | 3.0 | 6.1 | 2.3 | Iris-virginica |
| 49 | 5.0 | 3.3 | 1.4 | 0.2 | Iris-setosa |
| 89 | 5.5 | 2.5 | 4.0 | 1.3 | Iris-versicolor |

```
train_size = int(req_data.shape[0]*0.7)

train_df = req_data.iloc[:train_size,:]
test_df = req_data.iloc[train_size:,:]
train = train_df.values
test = test_df.values
y_true = test[:,-1]
print('Train_Shape: ',train_df.shape)
print('Test_Shape: ',test_df.shape)
```

```
Train_Shape:  (105, 5)
Test_Shape:  (45, 5)
```

```
from math import sqrt
def euclidean_distance(x_test, x_train):
    distance = 0
    for i in range(len(x_test)-1):
        distance += (x_test[i]-x_train[i])**2
    return sqrt(distance)

def get_neighbors(x_test, x_train, num_neighbors):
    distances = []
```

```
    data = []
    for i in x_train:
        distances.append(euclidean_distance(x_test,i))
        data.append(i)
    distances = np.array(distances)
    data = np.array(data)
      sort_indexes = distances.argsort() #argsort() function returns indices by sorting distances
data in ascending order
    data = data[sort_indexes] #modifying our data based on sorted indices, so that we can get the
nearest neighbors
   return data[:num_neighbors]

def prediction(x_test, x_train, num_neighbors):
    classes = []
   neighbors = get_neighbors(x_test, x_train, num_neighbors)
    for i in neighbors:
        classes.append(i[-1])
   predicted = max(classes, key=classes.count) #taking the most repeated class return
   predicted

def predict_classifier(x_test):
    classes = []
   neighbors = get_neighbors(x_test, req_data.values, 5)
    for i in neighbors:
        classes.append(i[-1])
   predicted = max(classes, key=classes.count)
   print(predicted)
   return predicted

def accuracy(y_true, y_pred):
   num_correct = 0
```

```
    for i in range(len(y_true)):
        if y_true[i]==y_pred[i]:
            num_correct+=1
    accuracy = num_correct/len(y_true)
    return accuracy


y_pred = []
for i in test:
    y_pred.append(prediction(i, train, 5))
y_pred
```

```
['Iris-virginica',
 'Iris-versicolor',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-setosa',
 'Iris-virginica',
 'Iris-virginica',
 'Iris-setosa',
 'Iris-versicolor',
 'Iris-virginica',
 'Iris-versicolor']
```

accuracy = accuracy(y_true, y_pred)

accuracy

```
0.9555555555555556
```

test_df.sample(5)

| | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|---|---|---|---|---|
| 113 | 5.7 | 2.5 | 5.0 | 2.0 | Iris-virginica |
| 125 | 7.2 | 3.2 | 6.0 | 1.8 | Iris-virginica |
| 149 | 5.9 | 3.0 | 5.1 | 1.8 | Iris-virginica |
| 94 | 5.6 | 2.7 | 4.2 | 1.3 | Iris-versicolor |
| 99 | 5.7 | 2.8 | 4.1 | 1.3 | Iris-versicolor |

**RESULT:-**

Thus the python program to implement the K-Means model has been successfully implemented and the results have been verified and analyzed.